Razvoj modela strojnog učenja za rješavanje zatvorenikove dileme korištenjem metoda pojačanog učenja

Kovačić, Leo Filip

Sveučilište
Sjever

Završni rad br. 1/RINF/2024

Razvoj Modela Strojnog Učenja za Rješavanje Zatvorenikove Dileme Korištenjem Metoda Pojačanog Učenja

Leo Filip Kovačić, 0285005816

Đurđevac, listopad 2024. godine.

Sveučilište
Sjever

Odjel za Računarstvo i Informatiku

Završni rad br. 1/RINF/2024

Razvoj Modela Strojnog Učenja za Rješavanje Zatvorenikove Dileme Korištenjem Metoda Pojačanog Učenja

Student

Leo Filip Kovačić, 0285005816

Mentor

doc.dr.sc. Tomislav Horvat, mag.ing.inf.et.comm.tech.

Đurđevac, listopad 2024. godine.

297
257892

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR-42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za računarstvo i informatiku

STUDIJ: Prijediplomski stručni studij Računarstvo i informatika

PRISTUPNIK: Leo Filip Kovačić | MATIČNI BROJ: 0285005816

DATUM: 4.4.2024. | KOLEGIJ: Strojno učenje i umjetna inteligencija

NASLOV RADA: Razvoj modela strojnog učenja za rješavanje zatvorenikove dileme korištenjem metoda pojačanog učenja

NASLOV RADA NA ENGL. JEZIKU: Developing a machine learning model for solving the prisoner's dilemma using Reinforcement Learning methods

MENTOR: Tomislav Horvat | ZVANJE: docent

ČLANOVI POVJERENSTVA:
1. doc.dr.sc. Domagoj Frank - predsjednik
2. Dražen Crčić, predavač - član
3. doc.dr.sc. Tomislav Horvat - mentor
4. mr.sc. Vladimir Stanisavljević, v. pred. - zamjenski član
5.

Zadatak završnog rada

BROJ: 1/RINF/2024

OPIS:

Zatvorenikova dilema klasični je problem teorije igara koji istražuje situaciju u kojoj se dva zatvorenika uhvaćena u zločinu suočavaju s izborom hoće li surađivati ili izdati jedan drugoga. Najbolji individualni rezultat zatvorenika postiže se izdajom drugog, dok se najbolji rezultat za obojicu dobiva suradnjom. Vođeni nesigurnošću i nevjerom, zatvorenik ima poticaj da izda drugoga, što vrlo vjerojatno dovodi do lošijih ishoda za obojicu. U radu će se koristiti originalni izvorni kod originalnog turnira te će se trenirati neuronska mreža, odnosno programirati model strojnog učenja za rješavanje zatvorenikove dileme pod varijabilnim setovima kriterija protivnika (dobronamjerne i zlonamjerne strategije kao i model strojnog učenja protiv samog sebe). Cilj rada razviti je strategiju koja se može dinamički prilagoditi okruženju te dati optimalan rezultat bez obzira na protivničku strategiju.
U radu je potrebno:
- istražiti područje primjene umjetne inteligencije u navedenom području te napraviti kratak pregled
- definirati i opisati pojam umjetne inteligencije uz naglasak na pojačano učenje, opisati konkretan problem koncepta nazvanog zatvorenikova dilema te načine prikupljanja podataka
- korištenjem programskog jezika Python, konkretno biblioteka tenserflow i jupyter, izraditi model strojnog učenja te skriptu za prikupljanje podataka
- napraviti eksperimentalnu evaluaciju dobivenih rezultate te rezultate usporediti s rezultatima ostalih istraživača

ZADATAK URUČEN: 22.5.2024. | POTPIS MENTORA:

SVEUČILIŠTE SJEVER

Predgovor

Iskreno mi je zadovoljstvo predstaviti temu iz dva dobro poznata područja koja se u moderno vrijeme često pojavljuju u kombinaciji. Htio bih se zahvaliti mentoru na vodstvu i strpljenju tokom izrade ovog rada. Također bih se htio zahvaliti obitelji i kolegama na pomoći i podršci iskazanoj tokom ovih godina studiranja.

Sažetak

Ovaj rad analizira kompleksnost Zatvorenikove dileme i istražuje različite strategije za rješavanje tog problema kroz primjenu strojnog učenja i umjetne inteligencije. U rada je razvijen n-dimenzionalni model koji omogućava optimizaciju strategija kroz vektorsku analizu te testiranje modela pomoću MLAgenta, algoritma strojnog učenja.

Kroz povijesni pregled Zatvorenikove dileme te ključnih strategija poput Tit-for-Tat (TFT), rad naglašava kako kontekst, duljina igre i struktura isplata utječu na uspješnost različitih strategija. Poseban naglasak stavljen je na "kingmaker" strategije, poput DOWNING-a, koje, iako same po sebi ne uspijevaju, značajno utječu na ishod drugih strategija.

Implementacija MLAgenta provedena je kako bi se istražila mogućnost strojnog učenja u razvoju optimalne strategije za Zatvorenikovu dilemu. U radu se razmatraju izazovi vezani uz podešavanje hiperparametara modela, poput veličine ulaznog sloja i funkcije nagrade.

U radu su istražene te kategorizirane neke prepreke čime se nastojala eliminirati pristranost kada se radi o odabiru strategija s obzirom na kompleksnost problema Zatvorenikove dileme, odrađeno treniranje agenta na temelju statičkih strategija te provedene simulacije u svrhu evaluacije istog.

Summary

This paper analyzes the complexity of the Prisoner's Dilemma and explores various strategies for solving this problem through machine learning and artificial intelligence. An n-dimensional model has been developed within the study, allowing for strategy optimization through vector analysis and testing of the model using MLAgent, a machine learning algorithm.

Through a historical overview of the Prisoner's Dilemma and key strategies such as Tit-for-Tat (TFT), the paper highlights how context, game length, and payout structure influence the success of different strategies. Special emphasis is placed on "kingmaker" strategies, such as DOWNING, which, while not successful on their own, significantly impact the outcome of other strategies.

The MLAgent implementation was conducted to investigate the potential of machine learning in developing an optimal strategy for the Prisoner's Dilemma. The paper discusses challenges related to tuning model hyperparameters, such as input layer size and reward function.

The study investigates and categorizes certain obstacles with the aim of eliminating bias in strategy selection regarding the complexity of the Prisoner's Dilemma. Agent training based on static strategies was performed, and simulations were conducted to evaluate the model.

Popis korištenih kratica

ZD	Zatvorenikova dilema
C	engl. “Cooperate” – hrv. Surađuj
D	engl. “Defect” hrv. – Izdaj
TFT	Tit-for-Tat
ML	engl. “Machine Learning” – Strojno učenje
AI	engl. “Artificial Intelligence” – Umjetna inteligencija

List of abbreviations

ZD	hrv. **“**Zatvorenikova dilema” – Prisoner’s Dilemma
C	Cooperate
D	Defect
TFT	Tit-for-Tat
ML	Machine Learning
AI	Artificial Intelligence

Sadržaj

1. Uvod 6
2. Prethodna istraživanja 9
3. Obrada zadatka 10

3.1. Kingmaker 11
3.2. DOWNING – popunjavanje praznine 12
3.3. Utjecaj duljine igre 13
3.4. Lekcije za dizajn mreže 15

4. Implementacija ZD u Python-u 17

4.1. Inicijalizacija igre i matrica nagrade 17
4.2. Igranje više rundi i povijest igre 18
4.3. Simulacija pojedinačne runde 19
4.4. Povijest i statistika igre 19
4.5. Implementacija MLAgent-a 20

5. Generiranje podataka za trening 21

5.1. RandomAgent 21
5.2. TitForTat 21
5.3. TitForTwoTats 22
5.4. TatForTit 22
5.5. JerkFace 22
5.6. MotherTheresa 23

6. MLAgent i njegove karakteristike 24

6.1. Struktura modela 24
6.2. Trening i adaptacija 24
6.3. Procjena učinkovitosti 26
6.4. Sposobnost učenja i dugoročna optimizacija 26

7. Rezultati 27
8. Zaključak 30
9. Literatura 32

1. Uvod

Zatvorenikova dilema[1] je misaoni eksperiment u teoriji igara koji uključuje dva racionalna agenta, od kojih svaki može surađivati radi međusobne koristi ili izdati svog partnera ("iznevjeriti") radi individualne nagrade. Ova dilema prvotno je postavljena od strane Merrilla Flooda i Melvina Dreshera 1950. dok su radili u RAND Corporation. Albert W. Tucker kasnije je formalizirao igru strukturiranjem nagrada u smislu zatvorskih kazni i nazvao je "zatvorenikova dilema" .

ZD modelira mnoge stvarne situacije koje uključuju strateško ponašanje. U svakodnevnoj upotrebi, oznaka "zatvorenikova dilema" može se primijeniti na bilo koju situaciju u kojoj bi dva entiteta mogli ostvariti važne koristi surađivanjem ili patiti zbog propuštanja istog, ali im je teško ili skupo koordinirati svoje aktivnosti.

William Poundstone opisuje je ovu "tipičnu suvremenu verziju" igre u svojoj knjizi "Prisoner's Dilemma" iz 1993. godine [2]:

"Dva člana kriminalne bande su uhićena i zatvorena. Svaki zatvorenik je u izolaciji, bez mogućnosti razgovora ili razmjene poruka s drugim zatvorenikom. Policija priznaje da nemaju dovoljno dokaza da osude par za glavni optužni prijedlog. Planiraju osuditi oboje na godinu dana zatvora za manju optužbu. Istovremeno, policija svakom zatvoreniku nudi faustovsku pogodbu. Ako svjedoči protiv svog partnera, bit će pušten na slobodu dok će partner dobiti tri godine zatvora za glavnu optužbu. Oh, da, postoji kvaka... Ako oba zatvorenika svjedoče jedan protiv drugog, obojica će biti osuđena na dvije godine zatvora. Zatvorenici imaju malo vremena da razmisle o tome, ali u svakom slučaju, ni jedan ne smije saznati što je drugi odlučio dok nije nepovratno donio svoju odluku. Svaki zatvorenik je obaviješten da se drugom zatvoreniku nudi ista takva ponuda. Svaki zatvorenik je zabrinut samo za vlastitu dobrobit - minimiziranje vlastite kazne zatvora.[2]*"*

Tradicionalni pristup rješavanja zatvorenikove dileme može se zamisliti kao dvodimenzionalna matrica koja predstavlja niz vektora koji prate poteze protivnika. U ovom kontekstu, svaki igrač ima mogućnost biranja između suradnje i izdaje, što se može predstaviti kao redak ili stupac u matrici. Matrica na taj način odražava rezultate suradnje ili izdaje oba igrača, prikazujući kako se kazne ili nagrade raspoređuju ovisno o odabiru oba sudionika.

Na primjer, redak matrice može predstavljati poteze prvog igrača (*C* - suradnja ili *D* - izdaja), a stupac poteze drugog igrača. Svaki element matrice odražava rezultat koji proizlazi iz kombinacije poteza oba igrača. Ovaj pristup omogućuje analizu različitih strategija i njihovih rezultata u igri za dileme, često koristeći tehnike poput teorije igara kako bi se identificirale optimalne strategije ili ravnoteže.

$$M = \begin{bmatrix} C\,C\,D \\ C\,D\,D \end{bmatrix}$$

Slika 1.1 Primjer matrice poteza u kojoj retci predstavljaju poteze pojedinog igrača

Zatvorenikova dilema, unatoč svojoj kompleksnosti, može se analizirati kroz prizmu vektorskog modeliranja kako bi se bolje razumjelo ponašanje igrača i optimizirale strategije. Svaki igrač može biti predstavljen kao vektor u *n*-dimenzionalnom prostoru, gdje su dimenzije određene različitim aspektima ponašanja ili preferencijama. Za početak, označimo ove vektore kao v_a i v_b za dva igrača. Zavisnost između članova jednog vektora naspram drugoga u ovom slučaju možemo smatrati strategijom.

$$v_a = [C\,,\,C\,,\,D];\; v_b = [C\,,\,D\,,\,D]$$

Slika 1.2 Vektori poteza

Iz takve tradicionalne reprezentacije je moguće izvući nagrade svake runde igre uz pomoć matrice nagrade, kao što je slijedeća.

$$N = \begin{bmatrix} [3,3] & [5,0] \\ [0,5] & [1,1] \end{bmatrix}$$

Slika 1.3 Matrica nagrada

Primjerice, u kontekstu teorije igara istražujemo različite ishode igre suradnje i izdaje među dvama igračima. Kada oba igrača surađuju, rezultat je nagrađujući za oboje, označen kao [3, 3]. Suprotno tome, ako jedan igrač surađuje dok drugi izdaje, rezultat je [5, 0], gdje izdajnik biva nagrađen više, dok suradnik trpi posljedice. U konačnici, u slučaju da oba igrača izdaju, rezultat je [1, 1], gdje oba igrača trpe negativne posljedice.

Analizom matrice igre, moguće je izvući nagrade za svaku rundu igre. Kroz matricu nagrada, dobivamo konkretne vrijednosti koje odražavaju ishode svakog mogućeg scenarija. U prethodno navedenom primjeru, dobivamo vrijednosti [3, 3], [5, 0] i [1, 1] zamjenom vrijednosti suradnje (*C*) i izdaje (*D*) u vrijednostima iz matrice nagrada *N*. Na taj način dobivamo numeričku reprezentaciju strategije, reflektirajući njenu učinkovitost kao nagrade za suradnju, izdaju ili kombinaciju oba poteza.

$$M = \begin{bmatrix} 3\,0\,1 \\ 3\,5\,1 \end{bmatrix}$$

Slika 1.4 Matrica igre s konkretnim vrijednostima nagrade

Kod ovakvog je prikaza svaki stupac matrice M jedan od članova matrice R. Dalje, vrijednosti unutar vektora v_a i v_b (C ili D) možemo zamijeniti numeričkim vrijednostima iz matrice nagrada. Ova reprezentacija omogućuje kvantifikaciju strategija i njihovih učinaka u prostoru teorije igara. Ukupna nagrada postaje ovisna o omjeru magnitude vektora i njegove dimenzionalnosti, pružajući jasnu mjeru uspjeha strategije unutar igre. Proširenjem originalne matrice igre M individualnim vektorima protivnika se dobiva matrica od n stupaca. Na taj način broj stupaca matrice odgovara dimenzionalnosti vektora v_a i v_b.

$$M = \begin{bmatrix} va_1 va_2 \dots va_n \\ vb_1 vb_2 \quad vb_n \end{bmatrix}; R_1(M) = v_a = [va_1, va_2, \dots, va_n]; R_2(M) = v_b = [vb_1, vb_2, \dots, vb_n]$$

$$(v_a, v_b) \in ((x, y) : (x, y) \in N)^n$$

Slika 1.5 Matrica igre i definicija vektora nagrade

Proširenjem matrice M u treću dimenziju dobivamo kontinuirani prostor svih mogućih strategija pod uvjetom da vektori v_a i v_b sadržavaju sve permutacije poteza. S obzirom da matrica nagrada sadrži 4 člana, veličina z dimenzije 3D matrice M je $4n$, gdje je n dimenzionalnost vektora v_a i v_b . Podaci prikupljeni iz stvarnih situacija ZD mogu se predstaviti kao skup vektora, od kojih svaki opisuje ponašanje igrača u određenoj situaciji. Kroz primjenu metoda strojnog učenja, modeli mogu automatski analizirati ove vektore, otkrivati obrasce ponašanja te optimizirati strategije.

Primjenom strojnog učenja na ove vektorske reprezentacije, modeli imaju priliku automatski prilagoditi svoje strategije, učeći iz ishoda prošlih rundi igre. Ovaj pristup omogućuje dublje razumijevanje mehanizama odlučivanja u Zatvorenikovoj dilemi i otvara vrata za optimizaciju strategija unutar zadanih okvira, čime se doprinosi širem polju teorije igara i primjene strojnog učenja.

2. Prethodna istraživanja

Temeljni rad Roberta Axelroda iz 1980. godine[3], koji je pionirski istražio različite strategije za Zatvorenikovu dilemu (ZD), osobito kroz organizaciju turnira, postavio je okvir za daljnja istraživanja u ovom području. Axelrodova strategija "Tit-for-Tat" (TFT) istaknula se kao dominantna zahvaljujući sposobnosti poticanja kooperacije kroz recipročno ponašanje. Međutim, kako je pokazao Axelrod, uspjeh strategije uvelike ovisi o kontekstu igre, uključujući duljinu interakcije i strukturu isplata.

Dodatno, Axelrod je uveo koncept "kingmaker" strategija, poput DOWNING-a, koje značajno utječu na uspjeh drugih strategija unatoč tome što same po sebi ne ostvaruju vrhunske rezultate.

Metode strojnog učenja, poput onih korištenih u ovom radu, mogu biti učinkovite, ali njihov uspjeh ovisi o pravilnoj definiciji okvira učenja i veličini podataka s kojima se agent suočava. Nasuprot tome, evolucijske strategije temelje se na dugoročnoj selekciji i adaptaciji, pružajući stabilnije rezultate u uvjetima dinamičnih i nepredvidivih okruženja[4]. Ova analiza učenja i evolucije podupire tezu da se različiti pristupi mogu koristiti za optimizaciju ponašanja agenata, ali je potrebna pažljiva implementacija ovisno o postavljenim parametrima igre.

U uvjetima u kojima igrači nemaju potpune informacije o potezima ili isplatama protivnika, postizanje optimalne strategije postaje iznimno zahtjevno. Nepotpune informacije otežavaju donošenje odluka i povećavaju neizvjesnost ishoda. Takvi uvjeti zahtijevaju visoku razinu prilagodljivosti agenata, koji moraju donositi odluke na temelju ograničenih i često nedosljednih podataka[5].

Ovi nedavni radovi[4][5] proširuju Axelrodovu teoriju naglašavajući ključnu ulogu fleksibilnosti i adaptacije agenata u promjenjivim uvjetima igre. Kombinacija strojnog učenja i evolucijskih pristupa, kao i sposobnost upravljanja informacijama u uvjetima nesigurnosti, može pružiti učinkovitiji okvir za razumijevanje i rješavanje složenih problema poput Zatvorenikove dileme u stvarnom svijetu. Stoga je ovaj rad, koji se oslanja na metodologiju strojnog učenja u pokušaju razvijanja optimalnih strategija za ZD, direktno povezan s ovim istraživanjima, naglašavajući potrebu za daljnjom optimizacijom i analizom implementiranih modela.

3. Obrada zadatka

Rad Roberta Axelroda iz 1980. godine, "Effective Choice in the Prisoner's Dilemma", ostaje ključno djelo u proučavanju suradnje u Zatvorenikovoj dilemi (ZD). U ovom radu, Axelrod predlaže niz eksperimentalnih turnira kako bi procijenio učinkovitost različitih strategija u ZD.

Pobjednik tih turnira je strategija nazvana Tit-for-Tat (TFT), koja surađuje u prvom potezu, a zatim oponaša prethodni potez protivnika u sljedećim potezima. Uspjeh TFT-a u Axelrodovim turnirima pripisan je kombinaciji "dobronamjernosti" i "opraštanja". Dobrota se odnosi na tendenciju suradnje čak i kada protivnik izda, dok opraštanje označava spremnost ponovnog uspostavljanja suradnje nakon privremenog izdajstva protivnika. Ove dvije osobine čine TFT posebno učinkovitom strategijom za poticanje suradnje u ZD, potičući protivnika na reciprocitet i odvraćajući ga od pretjerane izdaje.

Od Axelrodovih originalnih turnira provedena su istraživanja kako bi se dalje istražile osobitosti i implikacije Tit-for-Tat i drugih strategija u ZD. Ovo istraživanje ispitalo je različite čimbenike koji mogu utjecati na uspjeh različitih strategija, uključujući duljinu igre, troškove izdajanja i prisutnost drugih igrača[4][5][6].

Jedno od ključnih otkrića iz ovog istraživanja jest da je uspjeh TFT-a uvjetovan kontekstom u kojem se igra. Na primjer, pokazalo se da je TFT manje učinkovit u situacijama gdje je vjerojatnost buduće interakcije niska. To je zato što se TFT-ova priroda opraštanja može iskoristiti od strane igrača koji rano izdaju, a zatim ponovno izdaju kasnije.

Zaključci Axelrodovog rada ukazuju da je TFT općenito uspješna strategija u Zatvorenikovoj dilemi. Pokazala se boljom od različitih drugih strategija u raznim kontekstima. Uspjeh TFT-a ovisi o kontekstu u kojem se igra. Manje je učinkovita u situacijama gdje je vjerojatnost buduće interakcije niska ili gdje su isplate za suradnju i izdaju asimetrične. Druge strategije također mogu biti uspješne u Zatvorenikovoj dilemi, ovisno o određenom kontekstu[7]. Međutim, TFT je općenito dobra polazna točka za oblikovanje uspješne strategije. Axelrod također navodi dobronamjernost i oprostivost kao principe za uspješnu strategiju [3].

Axelrod također opisuje tzv. “kingmaker” strategije. To su strategije koje same nemaju dobar konačan rezultat, ali zbog svoje strategije odvajaju uspješne strategije od neuspješnih.

3.1. Kingmaker

U svome radu Axelrod razmatra učinkovitost različitih strategija u ZD, fokusirajući se na interakcije među različitim pravilima. Ključno opažanje je da na relativno rangiranje najboljih strategija značajno utječu dvije strategije, identificirane kao "kingmakeri", a to su GRAASKAMP i DOWNING [3].

DOWNING se smatra najvažnijim "kingmakerom" zbog širokog raspona rezultata kada je sparivan s drugim strategijama. Njegova logika donošenja odluka je posebna, jer pretpostavlja da suparnik temelji svoj odabir na prethodnom potezu DOWNING-a. DOWNING izračunava vjerojatnost da će suparnik surađivati nakon što DOWNING surađuje ili prekrši dogovor, a te procjene ažurira s svakim potezom. Mana u implementaciji DOWNING-a je početna pretpostavka o ne reaktivnom suparniku, što dovodi do ranih prekršaja i negativnog utjecaja na njegovu izvedbu. Međutim, ova mana čini DOWNING učinkovitim "kingmakerom", utječući na rangiranje drugih strategija.

Drugi "kingmaker", GRAASKAMP, koristi sofisticiranu strategiju kombinirajući TFT i povremene prekršaje. Njegov jedinstveni pristup omogućuje mu prepoznavanje vlastitog "blizanca" i prilagodbu ponašanja prema tome. Ukupna izvedba GRAASKAMP-a nije impresivna, ali služi kao "kingmaker", utječući na rangiranje drugih strategija.

Značaj ovih "kingmakera" leži u njihovom značajnom utjecaju na rangiranje najboljih programa na turniru. TFT dobro se ponaša s oba "kingmakera", konačno osvajajući turnir, dok TIDEMAN AND CHIERUZZI zauzimaju drugo mjesto. Rangiranje "nice" programa na turniru usko se poklapa s njihovom izvedbom s oba "kingmakera", osim za jedan izuzetak, NYDEGGER.

Tekst također naglašava da RANDOM nije "kingmaker" jer svi sudionici znaju da će RANDOM biti prisutan na turniru. Izvedba "nice" programa s RANDOM pokazuje negativan odnos, što ukazuje da dobro igranje s RANDOM ne nužno korelira s ukupnim uspjehom na turniru. Naglašava se važnost oprosta u pravilima donošenja odluka, koristeći primjer FRIEDMAN-a koji dobro igra s RANDOM-om, ali nedostatak oprosta pri suočavanju s prekršajima utječe na njegovu izvedbu s "kingmakerima".

Kasnije u tekstu, Axelrod navodi sljedeće: “Treće pravilo koje bi pobijedilo na turniru bila je blaga modifikacija pravila DOWNING. Da je DOWNING započeo s početnim uvjerenjima da će drugi igrači biti reaktivni umjesto ne reaktivni, i samo bi pravilo također pobijedilo, i to s velikom prednošću. Kingmaker bi mogao postati kralj. Početna vjerovanja DOWNING-a o drugim igračima bila su pesimistična. Ispostavilo se da bi optimizam u vezi njihove reaktivnosti

ne samo bio precizniji već bi također rezultirao i uspješnijom izvedbom. To bi rezultiralo prvim mjestom umjesto desetog mjesta."

3.2. DOWNING – popunjavanje praznine

Početni nedostatak poteza je izazov s kojim se suočavaju mnoge strategije u igrama poput ZD. Taktika koja se koristi u praksi može uključivati pretpostavke o ponašanju drugih igrača, s mogućnošću prilagodbe strategije kako se igra odvija. Ako uzmemo da određena strategija uzima zadnjih n poteza protivnika, potrebno je te inicijale vrijednosti postaviti na neku konstantu kako bi strategija mogla obaviti analizu. Nazovimo poteze koje strategija analizira međuspremnikom (engl. buffer) – b kojega možemo smatrati vektorom određene dimenzionalnosti.

$$b = [b_1, b_2, b_3, \ldots, b_n]$$

Slika 3.1 Međuspremnik kao vektor

Kod pravila DOWNING početan međuspremnik izgleda ovako:

$$b_{DOWNING} = [D, D]$$

Slika 3.2 DOWNING-ov međuspremnik

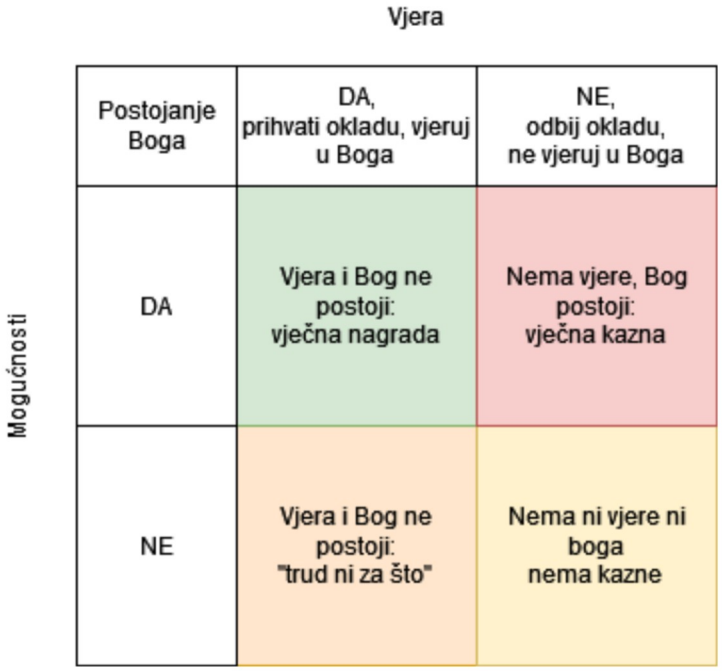
Kroz tijek igre, strategija postepeno zamjenjuje poteze u međuspremniku potezima protivnika na način da uklanja prvi član vektora (b_1 biva uklonjen, b_2 dolazi na mjesto b_1, b_3 na mjesto b_2 itd.) te na kraju na zadnje mjesto dodaje zadnji potez protivnika - v_{bi} gdje je i trenutna runda igre. Nakon n rundi, gdje je n veličina b, međuspremnik biva kompletno ispunjen potezima protivnika. "Greška" DOWNING-a je u tome što inicijalne vrijednosti popunjava izdajom pa u većini slučajeva i sam biva izdan na početku igre.

Axelrod navodi da se u daljnjem testiranju pokazalo da modifikacija DOWNING strategije gdje je međuspremnik ispunjen dvjema suradnjama se ponaša ne samo kao kingmaker, nego bi njegov prosječni rezultat na kraju turnira bio 542, što je znatno veće od pobjednika turnira kao i drugih predloženih strategija uključujući TFT. Također se spominje "ripple effect" prilikom kojeg jedna izdaja uzrokuje naizmjeničan val međusobnih izdaja, umanjujući konačan rezultat igre oba protivnika te na taj način i njihov konačan prosjek [3].

3.3. Utjecaj duljine igre

Prema općim zaključcima i dosadašnjim rezultatima analize ZD se pokazalo da duljina igre ima masivan utjecaj na vijabilnost strategija. Uzmimo na primjer najjednostavniju verziju ZD od samo jedne runde. Takva igra se svodi na jedan izbor. Primjer takve igre je Pascalova oklada:

"Pogledajmo ovu perspektivu i damo sljedeću izjavu: [...] Na koju stranu ćemo se nagnuti? Razum ne može odlučiti za nas ni na jednu ni na drugu stranu: s obje strane nas dijeli beskonačna provalija. Na kraju ove beskrajne provalije u tijeku je igra u kojoj se mogu pojaviti ili glava ili pismo. Na što ćete se kladiti? Prema razumu ne možete se kladiti ni na jednu ni na drugu stranu; prema razumu ne možete braniti nijednu od tih tvrdnji. Zato ne pripisujte grešku onima koji su napravili odabir; jer o tome ništa ne znate [8]." -Blaise Pascal – Les Pensées.



Slika 3.3 Pascal-ova Oklada

Primjenom konkretne matrice nagrada možemo navedeni primjer smatrati prvim zapisom ZD. Ako zatim konkretiziramo igru na dva protivnika, dobivamo sljedeću matricu:

$$N = \begin{bmatrix} [3,3] & [0,5] \\ [5,0] & [1,1] \end{bmatrix}$$

$$M = \begin{bmatrix} v_{a1} \\ v_{b1} \end{bmatrix}$$

Slika 3.4 Matrica nagrada i poteza – Modifikacija Pascal-ove oklade

Pascalov je argument da s obzirom na to da je protivnikov potez nepoznat, ne možemo birati redak u matrici nagrada nego samo stupac. Uzmimo prosječne očekivane vrijednosti za obje odluke; ako surađujemo, ili će protivnik surađivati (u kojem je slučaju nagrada 3), ili će nas izdati (u kojem je slučaju nagrada 0) u kojem je slučaju prosjek nagrada:

$$N_C = \frac{3+0}{2} = 1.5$$

Slika 3.5 Prosjek nagrade – C

U slučaju da protivnika izdamo, potencijalna je nagrada 5, dok je najmanja moguća nagrada.

$$N_D = \frac{5+1}{2} = 3$$

Slika 3.6 Prosjek nagrade – D

U ovakvom jednostavnom primjeru postoje samo dvije strategije, izdati ili surađivati – dobronamjerna ili zlobna koje možemo označiti kao v_C i v_D. Ako organiziramo turnir gdje svaka strategija igra sa svakom, v_C sama sa sobom uvijek ima rezultat (3, 3), v_C i v_D imaju (0,5), v_D i v_C imaju (5, 0), a v_D i v_D imaju (1, 1).

Iz toga možemo vidjeti da pobjednik igre odgovara očekivanim prosječnim rezultatima gdje je konačan prosječni rezultat:

$$\mu(v_C) = 3+0+\frac{0}{3} = 1\,; \mu(v_D) = 5+5+1 = 3.\overline{6}$$

Slika 3.7 Prosječne nagrade u igri jednog poteza

U konačnici, kod igre od jednog poteza, u prosjeku je racionalan izbor uvijek izdati protivnika. Nadalje, ako igru proširimo na dva poteza, postoje 4 moguće strategije koje možemo označiti prema njihovim potezima: v_{CC} , v_{CD} , v_{DC} i v_{DD}

Obradom rezultata takvog turnira dobivamo sljedeće: $v_{CC} = 24$, $v_{CD} = 36$,

$v_{DC} = 36$, $v_{DD} = 48$. Ako promatramo krajnje rezultate strategija kako broj rundi raste, najuspješnija će uvijek biti "uvijek izdaj", no moramo voditi računa da te strategije funkcioniraju samo u prostoru svih mogućih strategija te takav rezultat ne daje jasnu sliku o stvarnoj situaciji. Kod ovakvog, statičkog pristupa, strategije ne uzimaju u obzir poteze protivnika te se radi o svojevrsnoj "Babilonskoj Knjižnici" koja među mnoštvom naizgled nasumičnih zapisa također sadrži i sve smislene.

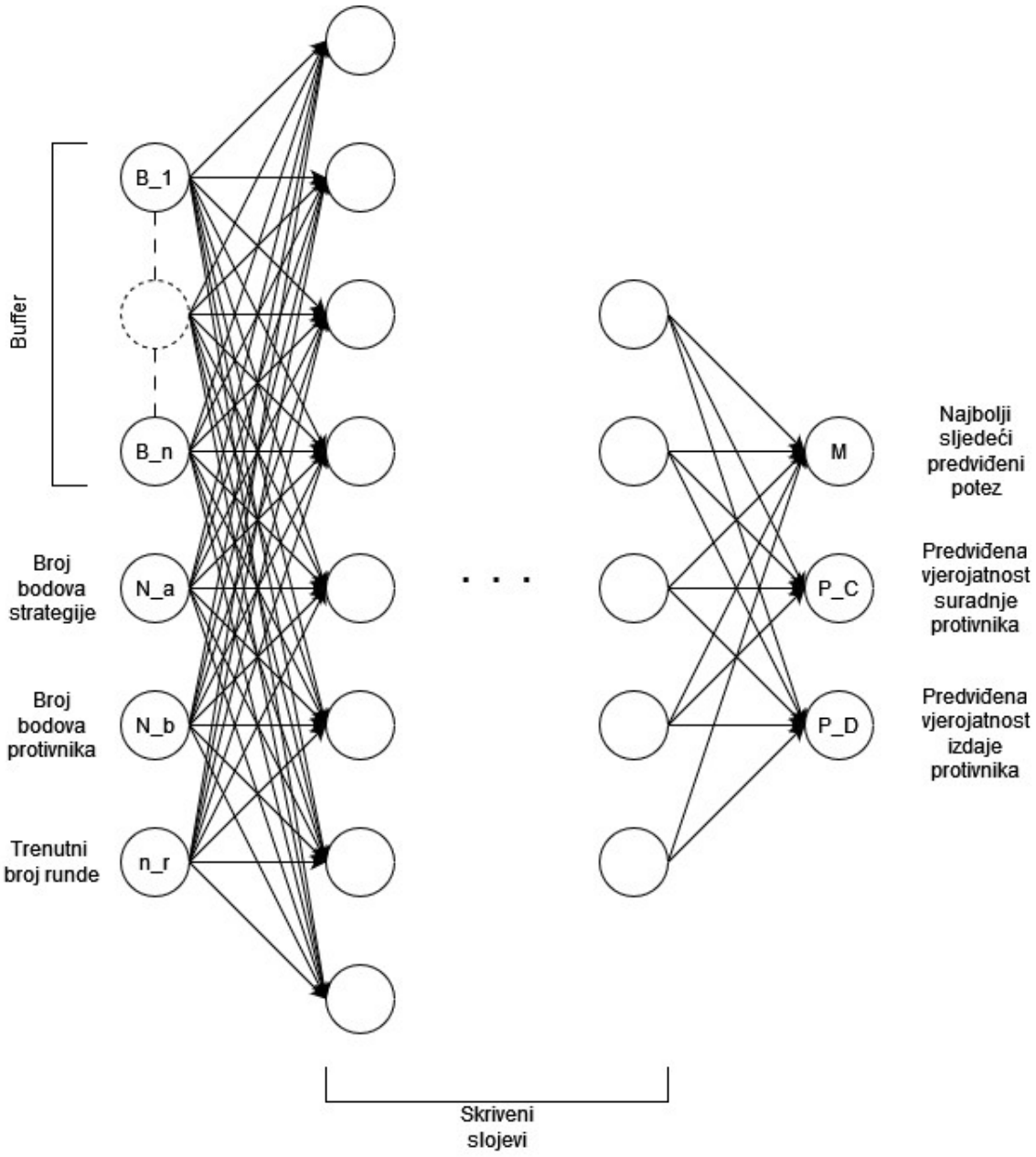
3.4. Lekcije za dizajn mreže

S obzirom na činjenice iz prethodnih poglavlja ovog rada, potencijalna neuronska mreža za rješavanje ZD bi trebala imati sljedeće karakteristike:

- dovoljno velik međuspremnik koji sadrži prethodne poteze protivnika
- međuspremnik inicijalno ispunjen pod pretpostavkom da su prethodni potezi protivnika bili C
- ulaz koji sadrži trenutni broj runde
- ulaz koji sadrži broj bodova neuronske mreže
- izlaz koji sadrži sljedeći potez neuronske mreže – *C* ili *D* kao broj između 0 i 1 (taj broj možemo smatrati vjerojatnošću isplativosti izdaje)
    - alternativno, tri izlaza od kojih prva dva predviđaju sljedeći potez protivnika dok treći daje optimalan potez neuronske mreže
- dovoljno velik broj skrivenih slojeva koji bi enkapsulirali kompleksnost zadatka

Vrijednost *C* i *D* je potrebno pretvoriti u binarni oblik gdje je *C=0*, a *D=1* što možemo smatrati vjerojatnošću izdaje. Vrijednost međuspremnika je uvijek cjelobrojna jer se radi o konkretnom prošlom događaju (vjerojatnost događaja koji se dogodio je 100%, dok je vjerojatnost događaja koji se nije 0%). Isto ne vrijedi za izlaze neuronske mreže koji mogu poprimiti raspon vrijednosti između 0 i 1 što predstavlja prostor vjerojatnosti budućih događaja.

Također, potrebno je odrediti niz strategija za trening i optimizaciju. ZD podrazumijeva da svaki protivnik ima iza sebe neku strategiju, u krajnju ruku strategija koja nasumično bira sljedeći potez. Važno je naglasiti da je prostor strategija jasno definiran za poznat broj rundi čistim izlistavanjem binarnih izbora, dok je cilj razviti strategiju gdje je taj broj nepoznat te potencijalno velik. U slučajevima gdje je broj rundi poznat, najbolja je strategija “uvijek izdaj” s obzirom na to da protivnici nemaju razloga surađivati prije toga. Iz tog razloga dolazi do rekurzije gdje protivnici znaju koji je protivnikov potez u zadnjoj rundi.



Slika 3.8 Dijagram neuronske mreže

4. Implementacija ZD u Python-u

Klasa PrisonersDilemma u game.py implementira iterativnu igru Dileme zatvorenika, gdje dva igrača kroz više rundi donose odluke hoće li surađivati (*C*) ili izdati (*D*). Ova klasa pruža osnovnu strukturu za pokretanje simulacije, definiranje povijesti igre, praćenje bodova i upravljanje ishodima svake runde na temelju unaprijed definirane matrice nagrade.

4.1. Inicijalizacija igre i matrica nagrade

U konstruktoru klase definirana je matrica nagrade koja sadrži isplate za sve moguće kombinacije poteza dvaju igrača. Inicijalizacija igre također uključuje varijablu history koja prati povijest poteza te total_scores koja bilježi ukupne bodove za svakog igrača.

```
class PrisonersDilemma:
    def __init__(self):
        # Payoff matrix for the Prisoner's Dilemma
        self.payoff_matrix = {
            ('C', 'C'): (3, 3),
            ('C', 'D'): (0, 5),
            ('D', 'C'): (5, 0),
            ('D', 'D'): (1, 1)
        }
        self.history = []
        self.total_scores = {'Player 1': 0, 'Player 2': 0}
```

Kod 4.1 Klasa PrisonersDilemma – matrica nagrada, povijest i bodovi

Matrica nagrade određuje ishode svake runde igre. Na primjer, ako oba igrača surađuju, svaki igrač dobije po 3 boda. Ako jedan igrač izda, a drugi surađuje, izdajnik dobiva 5 bodova, dok suradnik dobiva 0.

4.2. Igranje više rundi i povijest igre

Metoda play_game pokreće cijelu simulaciju igre za određeni broj rundi. Unutar svakog kruga, agenti donose svoje poteze pozivom na funkciju make_move, a nakon što se potezi donesu, poziva se metoda play_round koja simulira tu rundu, dodjeljujući nagrade igračima na temelju njihovih poteza.

```
def play_game(self, num_rounds, player1, player2):
  self.history = []
  self.total_scores = {'Player 1': 0, 'Player 2': 0}
  for _ in range(num_rounds):
    history = self.get_history()
    move1 = player1.make_move(history, 1)
    move2 = player2.make_move(history, 0)
    payoff1, payoff2 = self.play_round(move1, move2)
  history = self.get_history()
  print("\nGame History:")
   for round_num, (move1, move2, payoff) in enumerate(history,
start=1):
       print(f"Round {round_num}: \n\tPlayer 1 move: {move1},
\n\tPlayer 2 move: {move2}, \nPayoff: {payoff}\n")
  total_scores = self.get_total_scores()
  print("\nTotal Scores:")
  for player, score in total_scores.items():
    print(f"{player}: {score}")
```

Kod 4.2 Klasa PrisonersDilemma – metoda koja pokreće igru

U svakom krugu:

- igrači donose svoje odluke na temelju povijesti poteza, koju dohvaća funkcija get_history.
- zatim se poziva funkcija play_round, koja koristi matricu nagrade da izračuna isplatu za oba igrača.

Nakon svake runde, povijest poteza i nagrada se ažurira, čime se omogućuje retrospektivna analiza igre po završetku simulacije.

4.3. Simulacija pojedinačne runde

Metoda play_round simulira jednu rundu igre Dileme zatvorenika. Ova metoda uzima poteze oba igrača, primjenjuje matricu nagrade te ažurira povijest igre i ukupne bodove.

```
def play_round(self, move1, move2):
    payoff = self.payoff_matrix[(move1, move2)]
    self.history.append((move1, move2, payoff))
    self.total_scores['Player 1'] += payoff[0]
    self.total_scores['Player 2'] += payoff[1]
    return payoff
```

Kod 4.3 Klasa PrisonersDilemma – metoda za jednu rundu igre

Ova metoda:

- prvo dohvaća odgovarajuće nagrade iz matrice na temelju poteza igrača (move1, move2).
- zatim ažurira povijest igre (history), dodajući trenutne poteze i njihove isplate.
- ažurira ukupne bodove za oba igrača u varijabli total_scores.

Na kraju svake runde, metoda vraća isplate (payoffs) kako bi omogućila daljnju analizu ili prilagodbu strategije od strane agenata.

4.4. Povijest i statistika igre

Dodatne metode unutar klase PrisonersDilemma omogućuju dohvaćanje podataka o povijesti igre i ukupnim rezultatima igrača:

```
def get_history(self):
    return self.history
```

Kod 4.4 Klasa PrisonersDilemma – metoda za dohvaćanje povijesti igre

- **get_history** - vraća povijest igre, uključujući poteze igrača i njihove nagrade, što omogućava retrospektivnu analizu ponašanja agenata.
- **get_total_scores** - omogućuje pregled konačnih bodova nakon završetka igre.

Ove metode su ključne za evaluaciju uspjeha agenata tijekom simulacije, posebice kada se koristi MLAgent koji se prilagođava na temelju prethodnih poteza i nagrada.

4.5. Implementacija MLAgent-a

Jedan od ključnih elemenata u treniranju MLAgent-a jest funkcija nagrade, koja koristi matricu nagrade za izračunavanje isplata i prilagodbu modela tijekom simulacije. Funkcija train_round iz klase PrisonersDilemma koristi se za pokretanje rundi i prikupljanje podataka potrebnih za trening MLAgent-a.

```
def train_round(self, move1, move2):
   payoff = self.payoff_matrix[(move1, move2)] return payoff
```

Kod 4.5 Klasa PrisonersDilemma – klasa za treniranje naspram igranje runde

MLAgent koristi ovu funkciju kako bi prikupljao podatke o nagradama tijekom svake runde. Ove nagrade (payoff) potom se koriste za ažuriranje modela agenta putem reinforcement learning tehnika. funkcija train_round omogućuje agentu da trenira na temelju simuliranih igara, gdje na temelju povijesti poteza agent uči kako maksimizirati svoj ukupni rezultat.

Primjerice, tijekom treninga, MLAgent će koristiti isplate (nagrade) dobivene iz ove funkcije kako bi prilagodio svoje neuronske mreže, minimalizirajući funkciju gubitka i optimizirajući predviđanje najboljeg poteza u budućim rundama. Svaka iteracija igre pruža povratnu informaciju koja se koristi za prilagodbu modela kroz proces stohastičke gradijentne optimizacije (Adam optimizator). Funkcija gubitka je implementirana kao binarna unakrsna entropija (*engl. binary crossentropy*).

5. Generiranje podataka za trening

Za treniranje neuronske mreže je uzeto par algoritamskih strategije protiv koje neuronska mreža igra ZD. Na taj način su podaci za treniranje neuronske mreže generirani kao niz enkodiranih poteza za svakog od igrača.

5.1. RandomAgent

RandomAgent donosi odluke nasumično, bez analize prošlih poteza ili uvida u ponašanje protivnika. Njegova metoda make_move koristi Pythonov modul random za izbor između suradnje i izdaje. Zbog potpuno nasumične prirode, ovaj agent služi kao jednostavni bazni model protiv kojeg se uspoređuju naprednije strategije .

```
def make_move(self, history, opponent_id):
  return random.choice(['C', 'D'])
```

Kod 5.1 Klasa RandomAgent – glavna metoda

5.2. TitForTat

TitForTat agent implementira klasičnu strategiju "oko za oko". Počinje suradnjom, a zatim u svakom potezu oponaša prethodni potez protivnika. Strategija je vrlo jednostavna, a u teoriji igara se pokazuje kao učinkovita jer potiče dugoročnu kooperaciju. Međutim, agent je ranjiv na strategije koje eksploatiraju kooperativne protivnike [4].

```
def make_move(self, history, opponent_id):
  if history:
    self.last_opponent_move = history[-1][opponent_id]
    return self.last_opponent_move
```

Kod 5.2 Klasa TitForTat – glavna metoda

5.3. TitForTwoTats

TitForTwoTats strategija donosi odluku o izdaji tek nakoj dvije uzastopne izdaje protivnika. Ova strategija omogućuje veću toleranciju u odnosu na klasičnu TitForTat. Ako protivnik povremeno izda, TitForTwoTats će i dalje ostati suradljiv, no ako izdaja postane uzastopna, agent odgovara izdajom [4].

```
def make_move(self, history, opponent_id):
  if history and history[-1][opponent_id] == 'D':
    self.defection_count += 1
  else:
    self.defection_count = 0
  return 'D' if self.defection_count >= 2 else 'C'
```

Kod 5.3 Klasa TitForTwoTats – glavna metoda

5.4. TatForTit

TatForTit je inverzija strategije TitForTat, gdje agent na suradnju odgovara izdajom, a na izdaju suradnjom. Ova strategija je eksperimentalna, u praksi beskorisna te se koristi kao način destabilizacije protivnika. Zbog ovakvog ponašanja, često rezultira niskim bodovima za oba igrača.

```
def make_move(self, history, opponent_id):
  if history:
    self.last_opponent_move = history[-1][opponent_id]
  return 'D' if self.last_opponent_move == 'C' else 'C'
```

Kod 5.4 Klasa TatForTit – glavna metoda

5.5. JerkFace

JerkFace je agent koji stalno izdaje bez obzira na ponašanje protivnika. Ova jednostavna, nefleksibilna strategija koristi se kako bi se testiralo ponašanje drugih agenata u najgorim uvjetima.

```
def make_move(self, history, opponent_id): return 'D'
```

Kod 5.5 Klasa JerkFace – glavna metoda

5.6. MotherTheresa

Nasuprot JerkFace agentu, **MotherTheresa** stalno surađuje bez obzira na ponašanje protivnika. Ova strategija simulira naivnog igrača koji se uvijek ponaša altruistički, što ga čini ranjivim na iskorištavajuće protivnike.

```
def make_move(self, history, opponent_id): return 'C'
```

Kod 5.6Klasa MotherTheresa – glavna metoda

6. MLAgent i njegove karakteristike

MLAgent je agent koji koristi metode strojnog učenja kako bi se prilagodio ponašanju protivnika i optimizirao vlastite poteze kroz iteracije igre. Razlika između MLAgenta i prethodno navedenih agenata je u njegovoj sposobnosti učenja iz povijesti igre i prilagođavanja strategije na temelju tog iskustva. Dok ostali agenti koriste fiksne ili deterministične strategije, MLAgent koristi podatke dobivene kroz interakcije kako bi se poboljšao i maksimizirao dugoročne isplate.

6.1. Struktura modela

MLAgent koristi **neuronsku mrežu** implementiranu u okviru TensorFlow biblioteke. Mreža se sastoji od nekoliko slojeva koji obrađuju povijest poteza iz prethodnih rundi i na temelju te povijesti predviđaju optimalni potez za sljedeću rundu.

Neuronska mreža sastoji se od sljedećih slojeva:

- **ulazni sloj (engl. input layer)** – Ulazi u mrežu su 16 prethodnih poteza protivnika, kodirani kao nizovi *C* i *D*. Ovi podaci se transformiraju u binarni oblik gdje je $C = 0$, a $D = 1$. Ulazni potezi protivnika odgovaraju 16 ulaznih neurona. Ulazni sloj koristi **ReLu** kao funkciju aktivacije.
- **skriveni sloj (Hidden layer) –** Mreža ima jedan skriveni sloj sa 32 neurona i koristi **ReLU** kao funkciju aktivacije. Ovi slojevi uče nelinearne odnose između povijesti poteza i budućih odluka.
- **izlazni sloj (Output layer)** – Izlaz je binarna vrijednost koja predviđa hoće li agent surađivati ili izdati u sljedećoj rundi. Funkcija aktivacije izlaznog sloja je **sigmoidna**, koja daje vrijednost između 0 i 1. Ako je izlaz veći od 0.5, agent izdaje inače, surađuje.

Model je treniran pomoću **Adam optimizatora**, a koristi **binary crossentropy** kao funkciju gubitka. Ovi algoritmi omogućuju mreži da ažurira težine na temelju podataka prikupljenih tijekom igre, poboljšavajući time sposobnost donošenja odluka.

6.2. Trening i adaptacija

MLAgent se trenira kroz **samostalno igranje** protiv RandomAgent, TitForTat, TitForTwoTats, TatForTit, JerkFace i MotherTheresa strategija, što mu omogućuje da prepozna obrasce u protivnikovom ponašanju i prilagodi vlastitu strategiju. Tijekom svakog ciklusa treninga, agent se suočava s različitim strategijama i na temelju rezultata poboljšava svoj model.

Ključni elementi treninga su:

- **epochs** – agent igra više iteracija igara (epoha) kako bi naučio optimalne poteze, gdje svaka epoha predstavlja jedan kompletan ciklus treninga tijekom kojeg agent analizira podatke iz igre, ažurira svoj model i zatim se evaluira
- **rounds per game** – u svakoj igri agent se susreće s više rundi, prilikom čega prikuplja podatke o ponašanju protivnika gdje svaka runda pridonosi trenažnom skupu podataka za poboljšanje modela
- **games per epoch** – agent odigrava više igara unutar svake epohe kako bi naučio različite obrasce ponašanja

Treniranje je provedeno nasumičnim odabirom jedne od protivničkih strategija te simulacijom igre kroz 200 rundi, kroz 10 epoha s jednom igrom po epohi.

```
for x in range(10):
  opponent = random.choice(opponents)
  player.train(
    game=PrisonersDilemma(player, opponent),
    path="E:\workspace\prisoners_dilemma\ml_agent_model.h5",
    opponent=opponent,
    epochs=10,
    games_per_epoch=1,
    num_rounds=200
  )
```

Kod 6.1 main.py – trening

Unutar svake runde, MLAgent koristi funkciju `make_move`, koja se oslanja na predviđanje njegove neuronske mreže. Model procjenjuje povijest poteza, predviđa sljedeći potez protivnika i na temelju te predikcije odabire vlastiti potez.

```
def make_move(self, history, opponent_id):
  opponent_moves = [move[opponent_id]
    for move in history[-self.input_size:]]
      encoded_moves = self.encode_moves(opponent_moves)
      prediction = self.model.predict(np.array([encoded_moves]),
verbose=0)[0][0]
       return 'D' if prediction > 0.5 else 'C'
```

Kod 6.2 Klasa MLAgent – glavna metoda

Agent koristi povijest poteza iz prethodnih rundi kako bi predvidio sljedeći potez protivnika. Potezi su kodirani u binarni niz ($C = 0$, $D = 1$) prije nego što se proslijede neuronskoj mreži. Model predviđa vjerojatnost sljedećeg poteza protivnika te ako je predikcija veća od 0.5, agent će izdati; inače će surađivati.

6.3. Procjena učinkovitosti

Kako bi se procijenila učinkovitost MLAgenta, agent je odigrao ZD sa svakim od protivnika kao i protiv sebe samog. Simulacija je provedena dva puta, prva igra je trajala 100 rundi, a druga 500.

6.4. Sposobnost učenja i dugoročna optimizacija

Ključna prednost MLAgenta nad ostalim agentima jest njegova sposobnost dugoročnog prilagođavanja. Dok se TitForTat i ostale determinističke strategije oslanjaju na fiksna pravila, MLAgent stalno ažurira svoje ponašanje. Kako igre napreduju, agent može prepoznati složenije obrasce eksploatacije protivnika te se prilagoditi u cilju maksimiziranja bodova.

Primjena tehnika strojnog učenja omogućava MLAgentu da simulira sofisticirane strategije koje se prilagođavaju dinamičnim uvjetima igre. Ova sposobnost kontinuirane prilagodbe ključno je obilježje modernih ML sustava, koji nadilaze ograničenja tradicionalnih algoritama u domeni teorije igara.

7. Rezultati simulacija

U nastavku se nalaze tablice rezultata simulacija igara odigranih u svrhu evaluacije MLAgent-a. Rezultati su dobiveni simulacijom zatvorenikove dileme MLAgenta i strategija korištenih za njegovo treniranje. Simulacija je odrađena s 100 rundi po igri kao i 500 rundi po igri.

Stupci predstavljaju bodove dobivene primjenom matrice nagrada na poteze protivnika, koje je određena strategija dobila dok se njeni protivnici nalaze u retcima. Na dnu tablice je ukupan zbroj bodova strategije. U slučajevima kad strategija igra sama protiv sebe, ukupan rezultat obje iteracije strategije je zbrojen i prepolovljen kako bi se dobio prosjek. Na taj način se dobiva rezultat koji je usporediv sa ostalima (npr. ako TitForTat igra protiv sebe, obje iteracije dobiju 300 bodova tokom 100 rundi za pa ako se takav rezultat jednostavno zbroji on je duplo veći od rezultata protiv sličnih strategija kao što je TitForTwoTats).

Tablica 7.1 Rezultati simulacije ZD – 100 runda

	MLAgent	**RandomAgent**	**TitForTat**	**TitForTwoTats**	**TatForTit**	**JerkFace**	**MotherTheresa**
MLAgent	300	58	300	300	9	104	300
RandomAgent	307	224	232	196	232	324	144
TitForTat	300	234	300	300	225	104	300
TitForTwoTats	300	321	300	300	280	108	300
TatForTit	489	231	225	180	200	496	3
JerkFace	99	50	99	98	1	100	0
MotherTheresa	300	402	300	300	498	500	300

Ukupno:	2095	1520	1756	1674	1445	1736	1347

Tablica 7.2 Rezultati simulacije ZD – 500 runda

	MLAgent	RandomAgent	TitForTat	TitForTwoTats	TatForTit	JerkFace	MotherTheresa
MLAgent	1500	265	1500	1500	41	516	1500
RandomAgent	1496	1153.5	1142	913	1129	1448	780
TitForTat	1500	1117	1500	1500	1125	504	1500
TitForTwoTats	1500	1582	1500	1500	1400	508	1500
TatForTit	2467	1169	1125	900	1000	2496	3
JerkFace	493	248	499	498	1	500	0
MotherTheresa	1500	1996	1500	1500	2498	2500	1500

Ukupno:	10456	7530.5	8766	8311	7194	8472	6783

Uočeno je da agenti temeljeni na fiksnim pravilima, poput TitForTat i TitForTwoTats, uspješno potiču kooperaciju u ponavljajućim interakcijama, no njihova inherentna ranjivost na eksploataciju ukazuje na ograničenja deterministčkih strategija. Strategija TitForTat uvijek ima isti ili sličan rezultat strategije protivnika. Primjer ovakve ranjivosti su njihovi relativno slabi rezultati protiv strategije TatForTit koja se u ovom slučaju može smatrati „kingmaker“ strategijom. U testnom je okruženju MLAgent pokazao uvjetnu sposobnost adaptacije na testirane strategije. MLAgent se protiv dobronamjernih strategija sam ponaša kao dobronamjerna strategija i u simulacijama nikad ne izdaje prvi.

MLAgent ima slične performanse u obje simulacije. Uočeno je da kada MLAgent igra protiv dobronamjernih strategija obje strategije jednostavno surađuju. Rezultati takvih interakcija donose oba protivnika 300 bodova u prvoj te 1500 bodova u drugoj simulaciji.

U simulacijama JerkFace i MotherTheresa predstavljaju ekstreme mogućeg broja bodova. JerkFace kao kompletno zlonamjerna strategija protiv MotherTheresa-e ostvaruje maksimum bodova u obje simulacije (500 bodova u prvoj i 2500 u drugoj) te protiv samog sebe ostvaruje maksimalan broj bodova moguć protiv strategije koja konzistentno izdaje (100 bodova u prvoj i 500 u drugoj simulaciji). Zanimljivo je također napomenuti da protiv strategija RandomAgent i TatForTit, MLAgent ima sličan rezultat kao JerkFace, tj. poprima dinamičnije karakteristike koje mu osiguravaju bolje performanse protiv istih u odnosu na TitForTat i TitForTwoTats.

MLAgent pokazao je najbolje ukupne rezultate u obje simulacije, ostvarivši 2095 bodova u 100 rundi i 10456 bodova u 500 rundi. Njegova sposobnost adaptacije i učenja iz ponašanja protivnika omogućava mu da postigne visoke rezultate protiv nepredvidivih strategija, kao što su

TatForTit i RandomAgent. Ipak, može imati poteškoće u potpunom iskorištavanju naivnih strategija poput MotherTheresa-e zbog nevoljkosti da prvi izda.

TitForTat se pokazao kao druga najbolja strategija, ostvarivši 1756 bodova u 100 rundi i 8766 bodova u 500 rundi. Ova strategija je stabilna i pouzdana u poticanju dugoročne suradnje i ostvaruje visoke rezultate protiv suradničkih strategija. Međutim, ranjiva je na eksploataciju od strane strategija koje koriste njenu suradničku prirodu, poput TatForTit-a. TitForTwoTats ostvaruje slične, te malo slabije rezultate kao TitForTat, s 1674 boda u 100 rundi i 8311 bodova u 500 rundi.

JerkFace je ostvario solidne rezultate, s 1736 bodova u 100 rundi i 8472 boda u 500 rundi. Ova strategija uspješno eksploatira protivnike koji ne uzvraćaju izdaju, poput MotherTheresa-e, ali gubi bodove protiv strategija koje se brzo prilagođavaju i reagiraju na izdaje, kao što su TitForTat i MLAgent.

TatForTit je postigao prosječne rezultate, s 1445 bodova u 100 rundi i 7194 boda u 500 rundi. Ova strategija djeluje kao "kingmaker" jer destabilizira i mijenja dinamiku turnira, ali često donosi loše rezultate zbog svoje nepredvidive prirode, osobito protiv strategija koje potiču suradnju.

MotherTheresa je imala najniže rezultate, ostvarivši 1347 bodova u 100 rundi i 6783 boda u 500 rundi. Ova strategija najbolje se ponaša protiv onih koji potiču suradnju, ali je ekstremno ranjiva na strategije koje iskorištavaju njenu konstantnu suradnju, poput JerkFace-a i TatForTit-a.

RandomAgent imao je srednje rezultate, ostvarivši 1520 bodova u 100 rundi i 7530.5 bodova u 500 rundi. Njegova nasumičnost čini ga nepredvidivim, što ponekad rezultira dobrim ishodima protiv strateških protivnika. Međutim, njegova nedosljednost čini ga inferiornim protiv strategija koje bolje reagiraju na obrasce.

Detaljna analiza povijesti igre MLAgent-a u budućim istraživanjima ima potencijal donošenja boljih statičkih strategija kao i pružanje boljeg uvida u optimalne strategije ZD.

8. Zaključak

Zatvorenikova dilema je kompleksan problem s nizom parametara koje je potrebno uzeti u obzir prilikom dizajna strategije, unatoč svojoj naizglednoj jednostavnosti. Razvijanje neuronske mreže za rješavanje ovog problema u sebi sadrži niz izazova u svim stadijima razvijanja takve mreže. Potrebno je naći te optimizirati parametre ulaznog sloja kao što su veličina ulaznog međuspremnika te njegovu ovisnost o broju rundi.

U ovom radu analizirane su implementacije različitih agenata u Dilemi zatvorenika, s posebnim naglaskom na razvoj i primjenu MLAgent-a, koji koristi metode strojnog učenja za optimizaciju donošenja odluka. Kroz analizu agentne strategije, razlike u ponašanju među klasičnim i naprednim agentima, kao i primjenu matrice nagrade unutar igre, postavljena su važna pitanja o prirodi međusobne suradnje i konkurencije u dinamičnim okruženjima. Takova analiza bi eliminirala ili u krajnju ruku umanjila ljudsku pristranost o donošenju odluka i zaključaka na temelju ekstrema između “dobronamjernih” i “zlobnih” strategija[7].

Prema rezultatima simulacijama, MLAgent osvaja prvo mjesto dok su strategije TitForTat i RandomAgent na udaljenom drugom i trećem mjestu. Razlika u broju bodova MLAgent-a u odnosu na ostale strategije se sastoji od brze adaptacije i promjene ponašanja strategije nakon prve izdaje njegovog protivnika što mu daje prednost u odnosu na statičke strategije kao što su JerFace i TatForTit. Također, MLAgent protiv strategije RandomAgent postiže veoma slične bodove kao one postignute suradnjom sa “dobronamjernim” strategijama dok je rezultat RandomAgent-a protiv “dobronamjernih” strategija relativno nizak za oba protivnika.

Strategija TatForTit se u simulacijama ponaša u skladu s očekivanjima te djeluje kao “kingmaker” strategija zbog svoje nestabilnosti. Ona sama ne osvaja dobar rezultat, nego smanjuje rezultat “dobronamjernih” strategija kao što su TitForTat, TitForTwoTats i MotherTheresa.

Treniranje bi bilo poželjno obaviti kroz niz duljina rundi po igri kako bi se smanjio utjecaj duljine runde na ishod igre. Dodatna istraživanja su potrebna kako bi se smanjio utjecaj duljine igre na performanse strategija, s ponovljenim simulacijama i dodatnim treniranjem i uključivanjem većeg broja statičkih strategija kako bi se dobio širi statistički pregled rezultata te na temelju istih donijeli snažniji zaključci o performansama MLAgent strategije.

Rezultati ovog rada ukazuju na postojanje strategije koja je sposobna dinamički prilagoditi svoje ponašanje ovisno o protivniku te nadmašiti klasične algoritamske strategije u njihovoj učinkovitosti. Nadalje, ovi nalazi doprinose razumijevanju složenih strategija u teoriji igara i

pružaju osnove za daljnja istraživanja u području umjetne inteligencije i strojnog učenja. Buduće istraživačke inicijative mogle bi istražiti primjenu sličnih modela u složenijim dinamičkim okruženjima, uključujući simulacije u stvarnom svijetu, gdje se strategije i odluke često moraju prilagođavati promjenjivim uvjetima i nepredvidivim ponašanjima drugih sudionika.

Konačno, integracija metoda strojnog učenja s teorijom igara otvara nove horizonte za razvoj inteligentnih agenata, koji mogu učinkovito surađivati ili konkurirati u raznim društvenim i ekonomskim kontekstima, pružajući alate za analizu i modeliranje složenih interakcija u stvarnom svijetu.

9. Literatura

[1] Gill, M. B., Shaftesbury, Lord, Morris, W. E., Brown, C. R., & Hume, D. (1981). The Prisoner's Dilemma. *Debatable Philosophical Wisdom*, 170.

[2] Poundstone, W. (1993). *Prisoner's Dilemma*. Doubleday.

[3] Axelrod, R. (1980). Effective choice in the prisoner's dilemma. *Journal of conflict resolution*, *24*(1), 3–25.

[4] Hingston, P., & Kendall, G. (2004). Learning versus evolution in iterated prisoner's dilemma. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)* (Vol. 1, pp. 364–372). IEEE.

[5] Deng, X., & Deng, J. (2015). A study of prisoner's dilemma game model with incomplete information. *Mathematical Problems in Engineering*, *2015*(1), 452042.

[6] Axelrod, R., & Hamilton, W. D. (1981). The evolution of cooperation. *science*, *211*(4489), 1390–1396.

[7] Milgrom, P. R. (1984). *Axelrod's" The Evolution of Cooperation"*. JSTOR.

[8] Pascal, B. (1999). *Pensées and Other Writings*. Oxford University Press.

Popis slika

Slika 1.1 Primjer matrice poteza u kojoj retci predstavljaju poteze pojedinog igrača 5
Slika 1.2 Vektori poteza 6
Slika 1.3 Matrica nagrada 6
Slika 1.4 Matrica igre s konkretnim vrijednostima nagrade 6
Slika 1.5 Matrica igre i definicja vektora nagrade 7
Slika 3.1 Međuspremink kao vektor 11
Slika 3.2 DOWNING-ov međuspremnik 11
Slika 3.3 Pascal-ova Oklada 12
Slika 3.4 Matrica nagrada i poteza – Modifikacija Pascal-ove oklade 12
Slika 3.5 Prosjek nagrade – C 13
Slika 3.6 Prosjek nagrade – D 13
Slika 3.7 Prosječne nagrade u igri jednog poteza 13
Slika 3.8 Dijagram neuronske mreže 15

Popis tablica

Tablica 7.1 Rezultati simulacije ZD – 100 runda 24
Tablica 7.2 Rezultati simulacije ZD – 500 runda 25

Popis kodova

Kod 4.1 Klasa PrisonersDilemma – matrica nagrada, povijest i bodovi 16
Kod 4.2 Klasa PrisonersDilemma – metoda koja pokreće igru 17
Kod 4.3 Klasa PrisonersDilemma – metoda za jednu rundu igre 17
Kod 4.4 Klasa PrisonersDilemma – metoda za dohvačanje povijesti igre 18
Kod 4.5 Klasa PrisonersDilemma – klasa za treniranje naspram igranje runde 18
Kod 5.1 Klasa RandomAgent – glavna metoda 19
Kod 5.2 Klasa TitForTat – glavna metoda 19
Kod 5.3 Klasa TitForTwoTats – glavna metoda 20
Kod 5.4 Klasa TatForTit – glavna metoda 20
Kod 5.5 Klasa JerkFace – glavna metoda 20
Kod 5.6Klasa MotherTheresa – glavna metoda 20
Kod 6.1 main.py – trening 22
Kod 6.2 Klasa MLAgent – glavna metoda 22

Prilozi

Link na github repozitorij: https://github.com/lfkovacic/prisoners_dilemma

Upute za postavljanje tensorflow okoline: https://www.tensorflow.org/tutorials

Sveučilište
Sjever

IZJAVA O AUTORSTVU

Završni/diplomski/specijalistički rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Leo Filip Kovačić _____ *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog/specijalističkog (*obrisati nepotrebno*) rada pod naslovom Razvoj modela stojnog ulenja za rješavanje zatvorenikove dilema korištenjem meroda pojačarog učenja _____ *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

L. Filip Kovačić
_____
(vlastoručni potpis)

Sukladno članku 58., 59. i 61. Zakona o visokom obrazovanju i znanstvenoj djelatnosti završne/diplomske/specijalističke radove sveučilišta su dužna objaviti u roku od 30 dana od dana obrane na nacionalnom repozitoriju odnosno repozitoriju visokog učilišta.

Sukladno članku 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje umjetnička djelatnost i visoko obrazovanje.